Wie beeinflusst eine asymmetrische Marktdatenlatenz zwischen zwei Handelsplattformen die Umsetzung von Arbitrage-Strategien im Hochfrequenzhandel?

Maria-Wächtler-Gymnasium Essen

Facharbeit im LK Informatik

betreut von Herr Großkamp

Schuljahr 2025/2026, Jg. Q1

vorgelegt von: Luan Ajeti

am 23.02.2026

Inhaltsverzeichnis

Inhaltsverzeichnis 2
1. Einleitung 3
1.1 Zielsetzung 3
1.2 Aufbau der Arbeit 4
2. Theoretische Grundlagen 4
2.1 Definition des Hochfrequenzhandels 4
2.2 Latenz-Arbitrage 5
2.3 Marktdatenlatenz 5
2.4 Websocket und REST 7
3. Empirische Analyse der Asymmetrischen Latenz 7
3.1 Aufbau der Testumgebung 7
3.1.1 Auswahl der Handelsplattformen 8
3.1.2 Hardware- und Netzwerkumgebung 8
3.1.3 Software und verwendete Bibliotheken 8
3.1.4 Sicherheitsmaßnahmen 9
3.2 Methodik der Datenerfassung 10
3.3 Trading Bot Implementierung 11
3.4 Ergebnisse 13
4. Fazit 16
4.1 Reicht Informationsvorsprung allein aus? 16
4.2 Existiert ein Latenz-Schwellenwert? 16
4.3 Wird Arbitrage unter retail Börsen Bedingungen instabil? 17
5. Literaturverzeichnis 18
6. Anhang 18

1. Einleitung

In dieser Facharbeit geht es um das Problem einer asymmetrischen Latenz von Marktdaten und ihre Auswirkungen auf die Arbitrage-Strategie im Hochfrequenzhandel. Die Idee der Facharbeit entwickelte sich, als beim Lesen eines Beitrags von CNBC las, dass die Investmentbank Goldman Sachs hundert Millionen Dollar investiert hat, um wenige Millisekunden beim Aktienhandel zu sparen[1]. Goldman Sachs ist kein Einzelfall. Alle Hedge-Fonds, Investmentbanken und Hochfrequenzhandel-Firmen setzen sich dafür ein, dieses universelle Problem der Latenz im Aktienhandel weitgehend zu minimieren, denn Geschwindigkeit ist höchste Priorität im Handel, da der Aktienmarkt auf einer Wettbewerbsstruktur aufgebaut ist, die dem schnelleren Käufer die Aktie zuschreibt[2].

1.1 Zielsetzung

Diese Studie ist ein Experiment, das beweisen soll, dass ein Trading Bot der mit einer Arbitrage-Strategie fungiert, eine reale empirische Analyse der Marktdatenlatenz durchführen kann, welche untersucht ab welchem Zeitpunkt und unter welchen Software-Bedingungen die Latenz Arbitrage Strategie nicht mehr zuverlässig und umsetzbar wird. Die Quintessenz liegt darin, dass man auf Basis empirischer Daten deduziert, ab wann der Informationsvorsprung einer schnellen Handelsplattform in der Praxis ihren Nutzen verliert. Im weiteren Verlauf sollen folgende Postulate wissenschaftlich geprüft werden:

1. Ein Informationsvorsprung reicht allein nicht aus, wenn die Ausführungslatenz zu hoch ist
2. Es existiert ein Latenz-Schwellenwert, ab dem Arbitrage unprofitabel wird
3. Arbitrage wird unter retail Börsen Bedingungen instabil

Im Fazit sollen diese Postulate erneut aufgegriffen und anschließend verifiziert oder falsifiziert werden.

[1] Vgl. CNBC: “Goldman Sachs is spending $100 million to shave milliseconds off stock trades”, online abrufbar unter: https://www.cnbc.com/2019/08/01/goldman-spending-100-million-to-shave-millisecondsoff-stock-trades.html, Stand 17.01.2026

[2] Vgl. The Quarterly Journal of Economics, Vol. 137, Issue 1, 2022, S. 493

1.2 Aufbau der Arbeit

Die Facharbeit gliedert sich in „theoretische Grundlagen“ und „die praktische Umsetzung“, d.h. das Programmieren eines Arbitrage-Trading-Bots und die anschließende empirische Analyse sowie das abschließende Fazit, welches die Studie zusammenfasst. Zunächst werden Hochfrequenzhandel, Arbitrage-Strategien und die Rolle der Marktdatenlatenz erläutert. Des Weiteren werden auch die für die Implementation des Trading-Bots genutzten Technologien: Python, WebSocket-API und REST-API erläutert.

Im praktischen Teil dieser Facharbeit wird ein Trading Bot implementiert, der die Arbitrage zwischen zwei unterschiedliche Handelsplattformen untersucht. Aufgrund der Verzögerung bei der Preisübertragung können sich Preisdiskrepanzen ergeben, die das Programm nutzen soll. Zudem wird der Trading Bot mit Bitcoin handeln, da der Markt volatiler ist als bei herkömmlichen Aktienmärkten und differenziertere Ergebnisse erwartet werden. Der Bot führt im Binance Testnet kleine Handelsplatzierungen (*0.001 BTC*) aus, sobald eine Preisdifferenz erkannt wird, die einem im Code festgelegten Wert gleicht. Gleichzeitig sollen relevante Daten erfasst und analysiert werden. Abschließend werden die Daten ausgewertet und im weiteren Verlauf auf die in **Kapitel 1.1** formulierten Postulate empirisch überprüft.

2. Theoretische Grundlagen

Im Folgenden sollen Die Begriffe Hochfrequenzhandel, Arbitrage, Arbitrage-Strategien, Marktdatenlatenz, die Rolle von Technologie und APIs definiert und erläutert werden. Ziel hierbei ist, die Zusammenhänge zwischen Geschwindigkeit, Informationsvorsprung und Handelsausführung darzustellen, sodass die theoretischen Grundlagen für das Verständnis der späteren Analyse, und der Implementation geschaffen werden.

2.1 Definition des Hochfrequenzhandels

In seiner Essenz fungiert der Hochfrequenzhandel bereits wie eine Form des Arbitragehandels, da versucht wird mit schnellem Zugriff auf News- und Data-Feeds mithilfe von Algorithmen die Fehlbewertungen und Ungleichgewichte von Wertpapieren zu analysieren und diese profitabel auszunutzen[3]. Anders als beim traditionellen Investor,

[3] Vgl. Seyfert, Robert: Der Quant. Zu einer neuen Figur der Ökonomie, S. 234

der auf einen Preisanstieg hofft, hofft ein HFT-Händler auch *Quant* genannt, explizit nur auf die Ungleichgewichte, die in der Gegenwart passieren oder auf positive probabilistische Ergebnisse aus der nahen Zukunft.

2.2 Latenz-Arbitrage

Es gibt diverse Arbitrage-Strategien, die im Rahmen des Hochfrequenzhandels existieren, doch diese Facharbeit konzentriert sich auf *Latenz-Arbitrage*. Diese Strategie tritt auf bei Diskrepanzen in der Ausführungsgeschwindigkeit und dem aktuellen Preis beim Aktienkauf. Dieser Vorteil wird ausgenutzt, mithilfe schneller Handelsaufträge und (Handels) Abbruch-Strategien, welche sich auf verzögerte Marktreaktionen berufen[4]. Verschiedene Börsen haben aufgrund physischer Entfernung oder softwarebedingten Schwachstellen unterschiedliche Latenzen, welche dazu führen, dass Preisinformationen nicht für jede Börse simultan gleich sind und Ziel der Strategie ist es, dementsprechend im quantitativen Maß auf der langsameren Börse zu handeln, bevor sie sich der schnelleren Angepasst hat.

Reguläre Marktteilnehmer sind zudem in diesem Rennen immens benachteiligt, aufgrund dessen, dass niedrige Latenzzeiten nicht erreichbar sind, ohne über fortgeschrittene technische Voraussetzungen zu verfügen wie: einen *Direct Market Access (DMA)* worüber HFT-Firmen ein direkter Zugang zum Marktplatz gewährt wird ohne eine Börse als Vermittler zu besitzen, was dementsprechend auch weniger Latenz bedeutet. Des Weiteren erfolgen HFT-Firmen über sog. *Co-Locations*, welche physische Nähe zu Börsen Servern suggerieren, indem man Plätze mietet, die sich in denselben Rechenzentren wie die Server der Börsen befinden[5].

2.3 Marktdatenlatenz

Unter Marktdatenlatenz versteht man die vergangene Zeit, die gebraucht wird, um über ein Ereignis, wie der Wechsel einer Handelsplatzierung zu lernen. Auf Basis dieser vergangenen Zeit wird im Anschluss eine Antwort generiert und der Handel zeitlich darauf

[4] Vgl. Alexander, Anthony. “Latency Arbitrage in Cryptocurrency Markets: Analyzing Execution Speeds & Liquidity Dynamics”: 2025

[5] Vgl. Stiftung Börse: "Techniken und Tools im High Frequency Trading (HFT)", online abrufbar unter: https://stiftung-boerse.de/tools/techniken-und-tools-im-high-frequency-trading-hft/, Stand 21.01.2026

basierend exekutiert [6] . Latenzen werden gewöhnlich in einem Zeitraum von Mikrosekunden gemessen. Bis vor einem Jahrzehnt wurde dieses Rennen an

Handelsaufträgen-Platzierungen auf der Ebene der Millisekunden gemessen, wobei heutzutage in Mikro- bis Nanosekunden gemessen wird[7].

Marktdatenlatenzen entstehen aufgrund der aktuellen Struktur des Markts, da Börsen herkömmlich mit dem *„First come, first served"* Prinzip operieren, wobei wissenschaftlich belegt wurde, dass dieser Vorteil nur gewährt wird auf Basis der Art und Weise wie *First Mover* (Pioniere) und *Follower* (Nachfolger) den Markt beeinflusst haben [8] . In einer anderen Studie erwies es sich, dass Pioniere, Marktführer werden und Branchenstandarts setzen. Die nachfolgenden Marktteilnehmer beeinflussen aufgrund ihrer Marktkonsolidierung und der Übernahme kausal an der Innovation im Wettbewerb Verhalten[9]. Diese Marktkonsolidierung (Senkung der Anzahl an Marktteilnehmern) gibt den Pionieren im Hochfrequenzhandel den Vorteil eine Dominanz im Markt zu gewinnen, indem man durch technologische Fortschritte und dementsprechender niedriger Latenz, dem Gewöhnlichen Folger einen Preis vorgibt mit jenen im weiteren Verlauf gehandelt wird. Für dieses Problem gibt es bereits ein Konzept zur Marktregulation, welche von der Börse IEX eingeführt wurde namens: *„Speed-Bump"*, die solch einen Vorteil weitgehend minimieren soll, indem man eine künstliche Verzögerung in der Handelsausführung implementiert, um Hochfrequenzhändler im Markt zu verlangsamen[10]. Ferner muss ein Begriff für die spätere Messung verstanden werden und das ist die sog.
„Paketumlaufzeit". Unter Paketumlaufzeit versteht man: *„[…] die Gesamtzeit, die benötigt wird, bis die Anfrage über das Netzwerk übertragen wird und die Antwort zurückgesendet wird."*[11]

[6] Vgl. Hasbrouck, Joel and Saar, Gideon. „Low-Latency Trading": 2013
[7] Vgl. Aquilina, Matteo, Eric Budish, and Peter O'Neill, "Quantifying the HighFrequency Trading 'Arms Race': A New Methodology and Estimates,": 2020.
[8] Vgl. Usero, Belén und Fernández, Zulima. „First come, first served: How market and non-market actions influence pioneer market share": 2009
[9] Vgl. Suntsova, Olesia. „THE IMPACT OF "FIRST-MOVER ADVANTAGE" ON GLOBAL MARKET STRUCTURE AND DYNAMICS": 2024
[10] Vgl. Woodward, Megan. „Bumping Up the Competition: The Influence of IEX's Speed Bump in US Financial Markets": 2018
[11] Vgl. Amazon AWS: „Was ist RTT im Netzwerk?", online abrufbar unter: https://aws.amazon.com/de/what-is/rtt-in-networking/, Stand 05.02.2026

2.4 Websocket und REST

In dieser Facharbeit wird die WebSocket-API für den kontinuierlichen Datenempfang genutzt, um preis Informationen mit möglichst geringer Latenz zu bekommen. WebSockets ermöglichen eine dauerhafte TCP-Verbindung zwischen einem Client und einem Server. Dadurch können Daten kontinuierlich und ohne wiederholten

Verbindungsaufbau übertragen werden. Studien zeigen, dass mit dieser Technik (abhängig von der Lokation) durchschnittliche Latenzen von etwa *30 Millisekunden* erreichbar sind[12]. Preis-Ticks und Order-Book-Aktualisierungen werden im Millisekunden Bereich empfangen, da nach dem Verbindungsaufbau ein kontinuierlicher Datenstrom entsteht. Über diesen Datenstrom ist eine bidirektionale Kommunikation möglich, d.h. Daten können über denselben Socket sowohl gesendet als auch empfangen werden[13]. Die REST-API wird für den Datenempfang und für die Handelsausführung verwendet. REST basiert auf dem *Request-Response-Prinzip*. Das bedeutet, dass Daten nur dann übertragen werden, wenn der Client eine Anfrage an den Server schickt.

3. Empirische Analyse der Asymmetrischen Latenz

Dieses Kapitel stellt den praktischen Teil dieser Facharbeit dar. Im Folgenden sollen die entwickelte Testumgebung vorgestellt, die Methodik der Datenerfassung erläutert und die Implementierung vom Trading-Bot beschrieben werden. Ziel der empirischen Analyse ist es, reale Marktdatenlatenzen zwischen zwei Handelsplattformen zu messen, und diese analytisch zu untersuchen auf die drei aufgestellten Postulate (siehe Kapitel 1.1).

3.1 Aufbau der Testumgebung

Aufgrund technischer Limitationen dieser Facharbeit, kann eine reale quantitative Analyse nicht direkt stattfinden, dennoch wird mit der folgenden Testumgebung eine möglichst authentische Abbildung durch folgende Spezifikationen untersucht. Die Testumgebung besteht aus den folgenden Handelsplattformen, Hardware und Netzwerkbedingungen,

[12] Vgl. Pandey Bhaskar, Dr. Gill Pramod. „Enhancing Real-time Web Applications with WebSockets: Performance and Responsiveness“: 2024

[13] Vgl. Wallarm: „Websocket Vs REST API“, online abrufbar unter: https://www.wallarm.com/what/websocket-vs-rest-api, Stand 07.02.2026

genutzter Software und Bibliotheken und für das Risikomanagement die Sicherheitsvorkehrungen.

3.1.1 Auswahl der Handelsplattformen

Latenz Arbitrage kann nur funktionieren, wenn auf zwei verschiedenen Börsen gehandelt wird, die eine Latenz Differenz betragen, welche *> 0* ist. Für die Börse mit weniger Latenz, die dementsprechend schneller Preisinformationen senden kann wird Kraken genutzt, da sie eine gute WebSocket Unterstützung hergibt, womit eine kontinuierliche Verbindung bestehen kann und sowohl Orderbuch-Updates als auch Preisticks innerhalb von *<2 Millisekunden* (**siehe Kapitel 3.2**) vom Trading Bot empfangen werden können.

3.1.2 Hardware- und Netzwerkumgebung

Der Trading-Bot wurde auf folgender Infrastruktur betrieben: Als Server wird ein dedizierter physischer Rechner (*OMEN GT16-0372NG*) benutzt mit einem *Intel Core i714700F Prozessor*, *32GB RAM*, welcher auf dem Betriebssystem *Windows 11 (Version: 24H2 Build: 26100.7462)* operiert. Die Netzwerkumgebung basiert auf einer kabelgebundene 1 Gbit/s Ethernet Verbindung, da WLAN zu hohe *Jitter* erzeugt. Jitter ist in diesem Kontext die Verzögerung bei der Übertragung von Datenpaketen, worunter Drahtlose Netzwerke die Ursache für Jitter sein können, aufgrund der niedrigeren Übertragungsrate und Bandbreite[14] als bei einer Drahtverbindung wie Ethernet Kabel.

Zudem wird die Studie in Essen (NRW, Deutschland) durchgeführt, was eine Nähe zum DE-CIX Frankfurt (einer der größten Internetknotenpunkte Europas) suggeriert. Diese Lage ermöglicht eine adäquate Anbindung an europäische Server.

3.1.3 Software und verwendete Bibliotheken

Für die Implementierung des Trading Bots und der anschließenden Auswertung wurde ausschließlich *Python 3.10.9* verwendet. Python wurde gewählt, weil es ein adäquates Equilibrium schafft zwischen Geschwindigkeit, Lesbarkeit und umfangreichen Bibliotheken. Besonders für Datenanalysen und Visualisierung ist Python weit verbreitet. Zudem würde ein Trading Bot in C++ den Rahmen dieser Facharbeit überschreiten, weshalb ich mich ausschließlich nur für Python entschieden hab.

Bibliotheken für den Trading Bot:

[14] Vgl. IONOS: „Was ist Jitter? Definition und vorbeugende Maßnahmen“, online abrufbar unter: https://www.ionos.de/digitalguide/server/knowhow/jitter/, Stand 11.02.2026

Die Bibliotheken time und datetime wurden für die Zeitmessungen, Zeitpunkte in der CSV und den Pausen zwischen der API-Anfragen benutzt. Für die Verbindung zur Binance Testnet-API (Preisabfragen und Orders) wurde *binance.client* benutzt. Die Konstanten *SIDE_BUY, SIDE_SELL und ORDER_TYPE_MARKET* wurden von der *binance.enums* Bibliothek entnommen. Für die kontinuierliche Preisabfrage von Kraken wurde *websocket* benutzt. Zum Parsen der WebSocket Nachrichten von Kraken wurde *json* benutzt. Um den WebSocket in einem separaten Thread laufen zu lassen, damit der Hauptloop nicht blockiert wird wurde *threading* benutzt. Zum Schreiben der Trade Daten, d.h. *Zeitpunkt, Preise, Diff, Action, Profit* usw., in die *trades.csv* Datei wurde die Bibliothek *csv* verwendet.

Bibliotheken für die Auswertung der CSV-Datei und die Erstellung der Diagramme:

Für das Einlesen und Verarbeiten der CSV-Datei, Filtern von BUY- und SELL-Trades, Berechnung von Mittelwerten, Mediane, Summen usw., wurde *pandas* verwendet. Des Weiteren wurde für die Erstellung und das Design der Diagramme *matplotlib.pyplot* und seaborn verwendet. Zur richtigen Verarbeitung der Zeitstempel aus der CSV wurde *datetime* benutzt.

3.1.4 Sicherheitsmaßnahmen

Da der Trading Bot im Binance Testnet betrieben wird, wurden mehrere Schutzmechanismen implementiert um Risiken, Logikfehler und API-Belastung zu minimieren. Das Ziel war, den Bot robust und kontrollierbar zu machen, auch bei Verbindungsproblemen, Fehlern oder unerwarteten Preissprüngen. Gleichzeitig war der Handel im Testnet aus rechtlichen Gründen zwingend erforderlich, da ich als 17-Jähriger noch nicht mit echtem Kapital oder echten Kryptowährungen handeln darf. Das Testnet von Binance ermöglicht es, die Strategie realistisch zu testen, basierend auf aktuellen Daten, ohne gegen die Alters-Vorschriften von Binance und Deutschland zu verstoßen und ohne jegliches finanzielle Risiko einzugehen. Zu den wichtigsten Sicherheitsmaßnahmen gehören, dass das Trading Bot, maximal eine offene Position (*position_offen*) halten darf. Bevor ein neuer Kauf ausgeführt wird, wird überprüft, ob bereits eine Position besteht. Dadurch entsteht keine Anhäufung von Positionen und kein Risiko durch mehrere parallellaufenden Trades. Zudem ist die Handelsmenge auf *0.001 BTC* festgelegt. Jede Position wird mindestens *5 Sekunden* gehalten (*min_halte_zeit = 5.0*). Das verhindert redundantes Traden bei kurzen Spreads und reduziert außerdem die Wahrscheinlichkeit von Fehlentscheidungen. Ferner, verkauft der Trading Bot nur, wenn entweder das Profit Ziel von *0,012%* erreicht ist (*profit_ziel*) oder die Preisdifferenz fast geschlossen ist

(*exit_diff = 0.04%*) und die Mindesthalte Dauer vergangen ist. Das schützt zum einen vor Verkäufen mit wenig Profit und zum anderen gibt es dem Spread Zeit, sich anzupassen. Der Trading Bot hat einen automatisch implementierten Stopp, wo nach *7200 Sekunden* Laufzeit der Bot abbricht und nicht mehr weiter handelt, um die Auswertung nach einem Standard beurteilen zu können und damit die API-Nutzung begrenzt wird. Hinsichtlich der Fehlerbehandlung bei API-Aufrufen, ist sowohl die Preisabfrage als auch die Order Platzierung in einem *try-except-Block* implementiert. Bei Fehlern wie z.B. Timeout oder einem Verbindungsabbruch, wird der Fehler in der Konsole ausgegeben und der Trading Bot pausiert *5 Sekunden*, bevor er weiterläuft, wodurch der Trading Bot nicht abstürzt.

3.2 Methodik der Datenerfassung

Bevor eine Festlegung zur Methodik der Datenerfassung erfolgen kann, muss bewiesen werden, dass eine der zwei gewählten Börsen eine niedrigere Latenz hat als die andere Börse, damit die Latenz Arbitrage Strategie aus pragmatischer Sicht sinnvoll ist. Hierzu wurden an den beiden Börsen *Binance* und *Kraken*, einmal mit WebSocket hundert Anfragen im *2 Sekunden* Takt geschickt und gleicherweise auch mit REST (siehe *Binance_latenz_test.py* und *Kraken_latenz_test.py*). Die folgenden Ergebnisse zeigen eindeutig:

	Binance WebSocket	Binance REST	Kraken WebSocket	Kraken REST
Durchschnittliche Paketumlaufzeit	5.613 ms	281.881 ms	0.847 ms	107.530 ms
Minimum Paketumlaufzeit	0.463 ms	270.429 ms	0.527 ms	85.462 ms
Maximum Paketumlaufzeit	344.908 ms	725.673 ms	2.376 ms	255.042 ms
Median Paketumlaufzeit	0.880 ms	274.305ms	0.773 ms	94.592 ms

Den durchgeführten Messungen zufolge lässt sich erkennen, dass vom Standort Essen (NRW, Deutschland) aus die durchschnittliche Paketumlaufzeit mit WebSocket bei Kraken um *4,766 ms* geringer ist als die zu Binance. Für REST waren die durchschnittliche

Paketumlaufzeit zu einem älteren Zeitpunkt bei Binance, da von Binance im Vergleich zu Kraken *174,351 ms* länger gebraucht hat, um die Anfragen zu schicken.

Deshalb wird Kraken als schnellere Börse für die Preisinformationen Datenerfassung genutzt und Binance zum Anfragen, da sie vom Experiment Standort aus, die langsamere Börse ist.

Die Datenerfassung in dieser Facharbeit basiert auf folgenden Werten, die bei jedem Kauf und Verkauf protokolliert werden. Es wird die Preisdifferenz (diff) berechnet als Differenz zwischen dem Preis von Kraken und dem Preis von Binance:

$$Preis_differenz = (kraken_preis - binance_preis) / binance_preis \times 100$$

Diese Metrik zeigt den Informationsvorsprung durch die Latenz-Asymmetrie und dient als Grund zum Kaufen für den Trading Bot. Zudem wird die Haltezeit (*halte_zeit*) einer Position protokolliert, die die Zeit vom Kauf bis zum Verkauf misst. Sie beobachtet, wie lange der Trading Bot wartet, bis eine der Abbruchbedingungen eintritt (*profit_ziel* oder *preis_differenz*). Auch der Netto-Profit pro Trade wird berechnet, wobei der Gewinn/Verlust nach Abzug der Gebühren das Ergebnis ist.

Alle Werte werden anschließend mit *Zeitpunkt*, *Preisen*, *Action* (*BUY* und *SELL*), *Order-IDs* usw. In die Datei *daten.csv* geschrieben.

3.3 Trading Bot Implementierung

Der Trading Bot wurde in Python implementiert, um die Latenz Arbitrage Strategie zwischen Kraken und Binance auch in der Praxis zu testen. Kraken wird als schnelle Quelle für Preisinformationen über eine WebSocket Verbindung genutzt und Binance als langsamere Plattform für den eigentlichen Handel über die REST-API. Die Implementierung lässt sich in folgende Kategorien unterteilen: Konfiguration, globale Variablen, Protokoll Funktion für das Schreiben in die CSV-Datei, WebSocket Verbindung über Kraken, REST-Anfrage für Binance und die zentrale Handelslogik im Hauptloop.

Zuerst werden die Konfigurationsparameter festgelegt. Die Binance API Zugangsdaten (*binance_api_key* und *binance_api_secret*) ermöglichen den Zugriff auf das Testnet. Weitere Variablen, die festgelegt wurden: *eingangs_spread*, die Preisdifferenz wo ein Kauf ausgelöst wird. Der mindeste Profit (*0,012%*) bei einem Kauf wurde als *profit_ziel* festgelegt. Die Variable *exit_differenz* (*0,04%*) gilt als Wert, welcher bei einer niedrigeren Preisdifferenz (nach der Mindesthaltedauer) signalisiert, dass verkauft werden muss. Die

minimale Haltezeit wurde mit *min_halte_zeit* auf *5 Sekunden* für jede Position festgelegt. Als Feste Handelsmenge pro Trade wird die variable *quantitaet* auf *0.001 BTC* gesetzt.

Alle Handelsplatzierungen werden in die Datei daten.csv protokolliert. Die CSV enthält die Spalten: *Zeitstempel, Kraken_Preis, Binance_Preis, Differenz, Action, Quantität, Netto_Profit, Haltezeit, Kauf_Order_ID* und *Verkauf_Order_ID.*

Weitere Globale Variablen wie *kraken_preis* und *binance_preis*, speichern den aktuellen Kurspreis. Ob eine Position gerade offen ist, wird protokolliert mit *position_offen*. Der Kaufpreis wird in *kauf_preis* gespeichert, die Kauf Zeit in *kauf_zeit* und die Identifikationsnummer des Kaufs in *kauf_order_id*. Der Momentane Profit wird in *aktueller_profit* gespeichert.

Das Protokollieren erfolgt über die Funktion *handel_protokollieren*, die jeden Trade in die CSV-Datei schreibt, bei Kauf und Verkauf.

Für Kraken wird eine WebSocket Verbindung, zur zweiten Version der API (*wss://ws.kraken.com/v2*) hergestellt. In der *bei_nachricht* Funktion wird jede eingehende JSON-Nachricht geparst. Bei Ticker Updates für BTC/USDT wird der letzte gehandelte Preis („last") extrahiert und in kraken_preis gespeichert. Bei Verbindungsaufbau (*bei_verbindungsaufbau*) sendet der Bot eine Subscribe Nachricht für den Ticker Kanal. Der WebSocket läuft auf einem separaten Thread, damit der Hauptloop nicht blockiert wird.

Der Binance Preis wird auf Anfrage über die REST-API in der Funktion *get_binance_preis* abgefragt (*client.get_symbol_ticker* für *BTCUSDT*). Da REST kein kontinuierlichen Datenstrom hergibt wie WebSocket, erfolgen die Anfragen bei jeder schleifen Iteration.

Die Handelslogik befindet sich in einer While Schleife, die automatisch nach *7200 Sekunden* (*2 Stunden*) abbricht. Zunächst wird der aktuelle Binance Preis geholt. Sind beide Preise verfügbar berechnet der Trading Bot die prozentuale Differenz:

$$Preis_differenz = (kraken_preis - binance_preis)/binance_preis \times 100$$

Dieser Wert, sowie Informationen wie, ob die Position offen ist, wie lange der Bot schon läuft und der Aktuelle Profit, werden kontinuierlich in der Konsole ausgegeben.

Liegt keine offene Position vor und ist preis_differenz > eingangs_spread, wird ein Kauf getätigt (*SIDE_BUY*, *ORDER_TYPE_MARKET*, *quantitaet*). Kaufpreis, Kauf Zeit und OrderID werden gespeichert, *position_offen* auf True gesetzt und der Trade protokolliert.

Bei offener Position wird die Haltezeit in Sekunden (*haltedauer*) sowie der aktuelle Profit (*aktueller_profit)* berechnet.

Ein Verkauf erfolgt nur, wenn der Profit das *profit_ziel* erreicht hat oder die Differenz ≤ *exit_differenz* ist und die *min_halte_zeit* vergangen ist.

Dann wird ein Verkauf platziert (*SIDE_SELL*), der Bruttogewinn ermittelt, die Gebühren abgezogen, *position_offen* zurückgesetzt und alles in die CSV geschrieben.

Bevor die Schleife neu iteriert, pausiert der Trading Bot *1,2 Sekunden*, um die API nicht übermäßig zu belasten. Fehler wie z.B. Verbindungsprobleme oder Timeouts werden in *try-except-Blöcken* abgefangen. Bei Fehlern gibt es eine *5 Sekunden* Pause, sodass der Trading Bot stabil weiterlaufen kann. Nach Ablauf der *2 Stunden* stoppt der Trading bot automatisch.

3.4 Ergebnisse

Der Trading Bot wurde am *22.02.2026* für *2 Stunden* im Binance Testnet betrieben. Ziel war es Preisdifferenzen zwischen Kraken (*WebSocket, niedrige Latenz*) und Binance (*RESTAPI, höhere Latenz*) zu identifizieren und diese zu Handeln. Insgesamt wurden *312 Handelsplatzierungen* und Verkäufe ausgeführt. Die durchschnittliche Haltezeit einer Position betrug *7,1 Sekunden*, die Median-Haltezeit *5,8 Sekunden*. Die längste beobachtete Haltezeit lag bei *76,3 Sekunden*, die kürzeste bei *1,4 Sekunden*.

Die Ergebnisse zeigen, dass die Arbitrage-Strategie in toto negativ ausgefallen ist und nicht profitabel war. Der gesamte Netto-Profit über alle durchgeführten Trades beträgt 62,38 USDT. Damit entstand über den gesamten Handelszeitraum ein deutlicher Verlust.

Pro Trade ergab sich ein durchschnittlicher Verlust von *–0,1999 USDT*. Der Median lag mit *–0,2037 USDT* auf einem ähnlichen Niveau, was darauf hindeutet, dass die meisten Trades einen vergleichbaren negativen Betrag aufweisen.

Von insgesamt *312* durchgeführten Trades, hat ein einziger Trade seine Position mit einem positiven Netto-Profit geschlossen. Das entspricht einer Quote von nur *0,3 %*. Die Erfolgsrate der Strategie mit Anbetracht der Limitationen des Experiments ist somit gering.

Beim Einstieg in eine Position lag die durchschnittliche Preisdifferenz bei *+0,023 %*, d.h. dass der Preis bei der Börse Kraken im Durchschnitt höher war als bei Binance. Die maximal beobachtete Preisdifferenz beim Einstieg einer Position betrug *+0,230 %*, während die minimale Differenz *–0,006 %* betrug. Der Bot nutzte zwar kleine positive Spreads, welche

jedoch in den meisten Fällen nicht ausreichten um die Gebühren und Slippage zu Kompensieren.

Zusammenfassung der wichtigsten Ergebnisse:

Anzahl Round-Trips (Kauf + Verkauf)	*312*
Gesamter Netto-Profit	*-62,38 USDT*
Durchschnittlicher Profit pro Handel	*-0,1999 USDT*
Median Profit pro Trade	*-0,2037 USDT*
Gewinnquote (positive Trades)	*0,3 %*
Durchschnittliche Haltezeit	*7,1 Sekunden*
Median Haltezeit	*5,8 Sekunden*
Durchschnittliche Diff beim Einstieg	*+0,023*
Maximale Diff beim Einstieg	*+0,230*

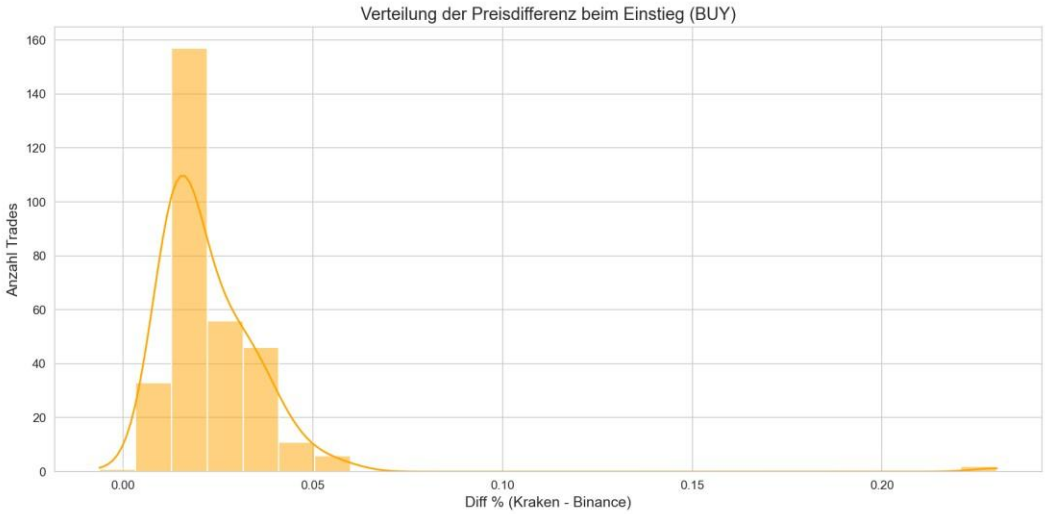

Abb. 1

Abbildung 1 zeigt die Verteilung der Preisdifferenzen beim Einstieg einer Position (*BUY*). Es ist erkennbar, dass die Mehrheit der Trades bei sehr geringen Differenzen (*0,01-0,05*) stattfand.

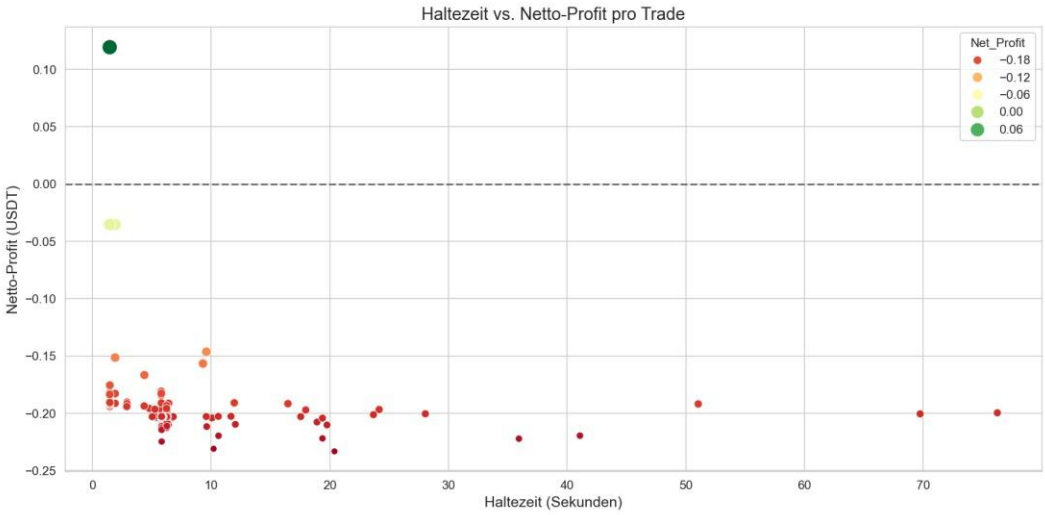

Abb. 2

Abbildung 2 visualisiert, dass nahezu alle Trades unter *0 USDT* liegen, unabhängig von ihrer Haltezeit, welche die inadäquate Rendite aufweist.

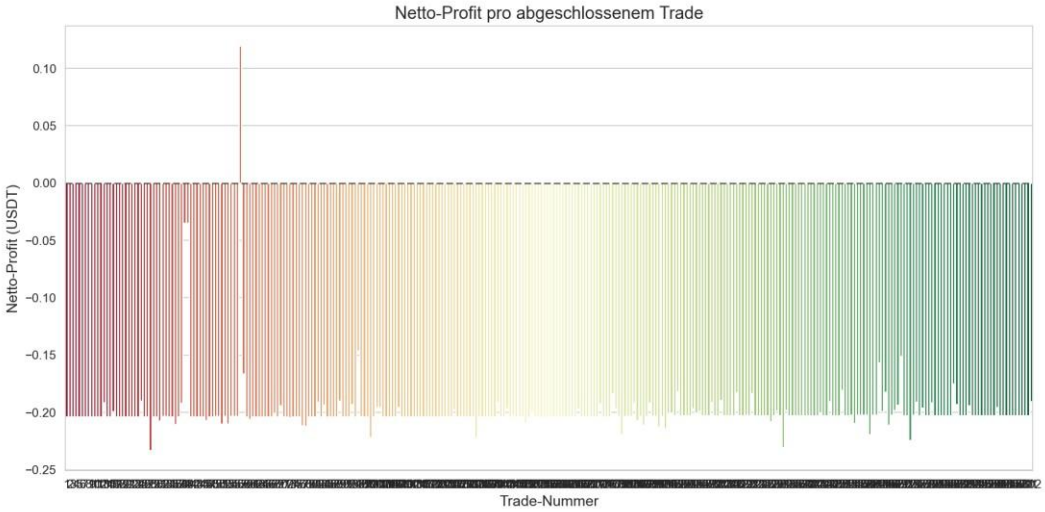

Abb. 3

Abbildung 3 zeigt den Netto-Profit pro Trade in chronologischer Reihenfolge. Der konstante negative Bereich zeigt wie in Abb. 2 die inadäquate Rendite der Arbitrage Strategie unter den gegebenen Bedingungen.

4. Fazit

4.1 Reicht Informationsvorsprung allein aus?

„Ein Informationsvorsprung reicht allein nicht aus, wenn die Ausführungslatenz zu hoch ist"

Obwohl der Trading Bot einen positiven Vorsprung hinsichtlich der Preisdifferenz nutzte, resultierten alle *312 Trades* in einem negativen Netto-Profit (*Durchschnitt –0,1999 USDT*). Der Informationsvorsprung durch die asymmetrische Latenz verlor durch Ausführungsverzögerungen, Slippage und Transaktionskosten seinen Nutzen beim Experiment. Dies zeigt, dass ein reiner Vorsprung in der Datenankunft ohne niedrige Ausführungslatenzen (wie bei professionellem HFT mit Co-Location) nicht ausreicht, um profitabel zu handeln.

4.2 Existiert ein Latenz-Schwellenwert?

„Es existiert ein Latenz-Schwellenwert"
Die Ergebnisse deuten auf einen Schwellenwert hin. Bei den beobachteten Kauf Latenzen (*Mittelwert nur +0,023 %, Maximum +0,230 %*) lagen die Netto-Profite durchgehend im negativen Bereich (*-0,20 USDT pro Handel*). Selbst die höchste beobachtete Differenz von 0,230 % reichte nicht aus, um die Gebühren (*0,15 % für Kauf + Verkauf*) und Preisveränderungen (*Slippage*) zu kompensieren. Der sog. *Break-even-Point*, also die minimale Preisdifferenz, wobei kein Verlust und kein Gewinn entsteht, lässt sich mit folgender Formel abschätzen:

$$Break\ Even\ Spread \approx Gebühren + Slippage + Puffer$$

$$0{,}25\text{-}0{,}35\% \approx 0{,}15\% + ca.\ 0{,}14\% + 0{,}05\text{-}0{,}10\%$$

Für die Gebühren müssen der Kauf und der Verkauf in Betracht gezogen werden, woraus für *2 x 0,075% = 0,15%* ergeben. Der Slippage wurde berechnet, indem man durch den durchschnittlichen Verlust mit dem Volumen pro trade von ca. *68 USDT* teilt, also ca. *0,144%* bei $\frac{-0{,}1999 USDT}{68 USDT}$. Aufgrund der Instabilität wurden noch *0,05-0,10%* dazu gerechnet, weil die Strategie sehr instabil war (Gewinnquote nur *0,3%* + schwankende Haltezeiten). Dieser Wert von *0,25-0,35%* ist also die Differenz, die man mindestens braucht, um die Kosten auszugleichen. In meinen 312 Trades kam so eine große Differenz fast nie vor und wenn doch, dann kurz. Aufgrund dessen war fast alles unprofitabel. Ohne

rapide Software und Hardware und niedrigen Gebühren rentiert sich Latenz Arbitrage bei kleinen Differenzen nicht.

4.3 Wird Arbitrage unter retail Börsen Bedingungen instabil?

„Arbitrage wird unter retail Börsen Bedingungen instabil“

Die Ergebnisse verifizieren, dass die Strategie unter den gegebenen Bedingungen instabil ist. Obwohl der Trading Bot klare Regeln hatte (Mindesthaltedauer von *5 Sekunden*, Exit bei *0.012% Profit* oder *Diff* $\leq$ *0,04%*), schwankten die Haltezeiten immens, von nur *1,4 Sekunden* bis zu *76,3 Sekunden*. Die durchschnittliche Haltezeit lag bei *7,1 Sekunden*, der Median bei *5,8 Sekunden*. Zudem lief der Trading Bot auf einem normalen Gaming-PC und einer 1-Gbit/s-Ethernet-Verbindung von zu Hause aus in Essen (NRW, Deutschland) (**siehe Kapitel 3.1.2**). Auch wenn Essen nah am DE-CIX in Frankfurt liegt, ist die Latenz und Stabilität nicht mit einem echten Hochfrequenzhandel Setup vergleichbar. Professionelle HFT-Firmen nutzen Co-Location (Server direkt im Rechenzentrum der Börse), spezielle Hardware wie FPGA (*Field-Programmable Gate Arrays*), welche speziell angefertigt werden für den quantitativen Handel mit möglichst niedriger Latenz. Durch die Einschränkungen dieses Experiments, kamen kleine Verzögerungen (*Jitter*) und Preisänderungen schneller hinzu, als der Trading Bot reagieren konnte. Des Weiteren konnte der Trading Bot den Vorsprung nicht nutzen, bevor der Spread sich schloss oder sogar umkehrte, was den hohen Verlust begründet.

Zusammenfassend lässt sich festhalten, dass Latenz-Arbitrage im Hochfrequenzhandel zwar theoretisch auf einem Informationsvorsprung basiert, in der Praxis jedoch nur mit fortgeschrittener technischer Ausstattung profitabel bleibt. Die Ergebnisse verifizieren somit alle drei Postulate: Ein Informationsvorsprung reicht bei hoher Ausführungslatenz nicht aus, ein Schwellenwert für profitable Arbitrage liegt unter den Bedingungen des Experiments bei *0,25* bis *0,35%*, und die Strategie erweist sich in einem technisch mangelnden Setup, als instabil und unprofitabel.

5. Literaturverzeichnis

1. Vgl. CNBC: “Goldman Sachs is spending $100 million to shave milliseconds off stock trades”, online abrufbar unter: https://www.cnbc.com/2019/08/01/goldman-spending-100-million-to-shavemilliseconds-off-stock-trades.html, Stand 17.01.2026
2. Vgl. The Quarterly Journal of Economics, Vol. 137, Issue 1, 2022, S. 493–564 3. Vgl. Seyfert, Robert: Der Quant. Zu einer neuen Figur der Ökonomie, S. 234
4. Vgl. Alexander, Anthony. “Latency Arbitrage in Cryptocurrency Markets: Analyzing Execution Speeds & Liquidity Dynamics”: 2025
5. Vgl. Stiftung Börse: "Techniken und Tools im High Frequency Trading (HFT)", online abrufbar unter: https://stiftung-boerse.de/tools/techniken-und-tools-imhigh-frequency-trading-hft/, Stand 21.01.2026
6. Vgl. Hasbrouck, Joel and Saar, Gideon. „Low-Latency Trading“: 2013
7. Vgl. Aquilina, Matteo, Eric Budish, and Peter O’Neill, “Quantifying the HighFrequency Trading ‘Arms Race’: A New Methodology and Estimates,”: 2020
8. Vgl. Usero, Belén und Fernández, Zulima. „First come, first served: How market and non-market actions influence pioneer market share“: 2009
9. Vgl. Suntsova, Olesia. „THE IMPACT OF “FIRST-MOVER ADVANTAGE” ON GLOBAL MARKET STRUCTURE AND DYNAMICS“: 2024
10. Vgl. Woodward, Megan. „Bumping Up the Competition: The Influence of IEX's Speed Bump in US Financial Markets“: 2018
11. Vgl. Amazon AWS: „Was ist RTT im Netzwerk?”, online abrufbar unter: https://aws.amazon.com/de/what-is/rtt-in-networking/, Stand 05.02.2026
12. Vgl. Pandey Bhaskar, Dr. Gill Pramod. „Enhancing Real-time Web Applications with WebSockets: Performance and Responsiveness“: 2024
13. Vgl. Wallarm: „Websocket Vs REST API“, online abrufbar unter: https://www.wallarm.com/what/websocket-vs-rest-api, Stand 07.02.2026
14. Vgl. IONOS: „Was ist Jitter? Definition und vorbeugende Maßnahmen“, online abrufbar unter: https://www.ionos.de/digitalguide/server/knowhow/jitter/, Stand 11.02.2026

6. Anhang

Der Code für den Trading Bot, die Datenauswertung und Visualisierung, die Latenzmessung der beiden Börsen, die CSV-Datei und die generierten Diagramme lassen sich im folgenden GitHub Repository finden:

https://github.com/Luanigi/Informatik-LK-Facharbeit-Luan-Ajeti-2026